G.Karthick PRN: 22070122062

Mini Project Report

Title: ***Comparative Analysis of Cloud Scheduling Algorithms with Energy Profiling using CloudSim***

Problem Statement

In cloud computing environments, efficient resource scheduling is crucial to optimize performance and energy consumption. Common scheduling algorithms like First-Come First-Serve (FCFS), Shortest Job First (SJF), and Round Robin (RR) behave differently under various workloads. However, many existing simulations lack realism in terms of task durations, VM configuration, and energy tracking.

This project aims to simulate and compare different VM scheduling algorithms using CloudSim, with a focus on energy efficiency, makespan, and average task execution time. The goal is to provide realistic and insightful evaluation by enhancing CloudSim with energy tracking, multi-host support, and visualization capabilities.

Objectives:

- Implement and compare RR, FCFS, and SJF scheduling algorithms using CloudSim
- Integrate realistic cloudlet durations and a fixed number of cloudlets (100) to ensure statistically significant comparisons
- Track and report total energy consumption, makespan, and average execution time
- Support multi-host data centers and visualize energy and performance metrics
- Design and implement a CI/CD pipeline for automated deployment and testing

Implementation Overview

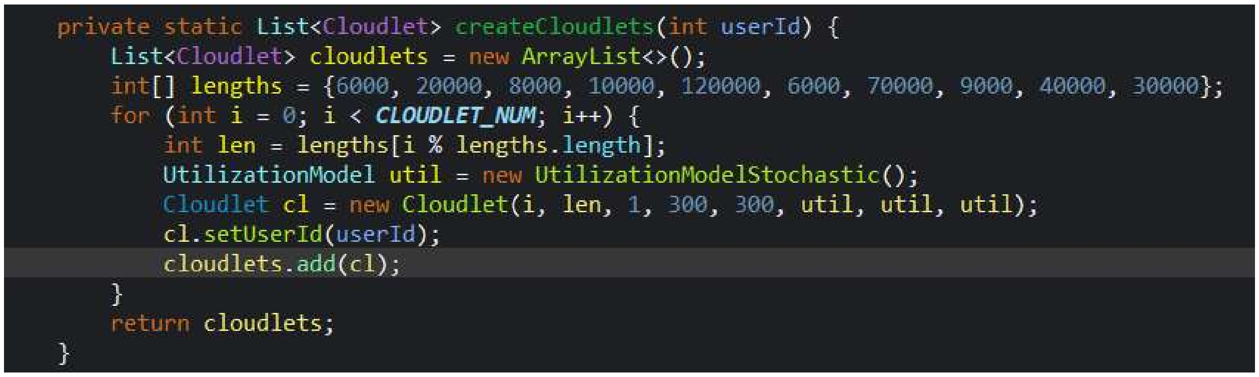
Simulation Design:

Component	Details
Simulator Framework	CloudSim 3.0.3
Language	Java
Power Model	Custom energy-aware extension
Scheduler Types	TimeShared (RR), SpaceShared (SJF, FCFS)
Hosts Configure	4 PEs @ 2 GHz 8 GB RAM 10 GB/s bandwidth Each host logs individual energy usage.
Host Count	Configurable (3 and 5 used in test cases)

Number of Cloudlets	**100**
Schedulers	**RR** → `CloudletSchedulerTimeShared` **FCFS, SJF** → `CloudletSchedulerSpaceShared`

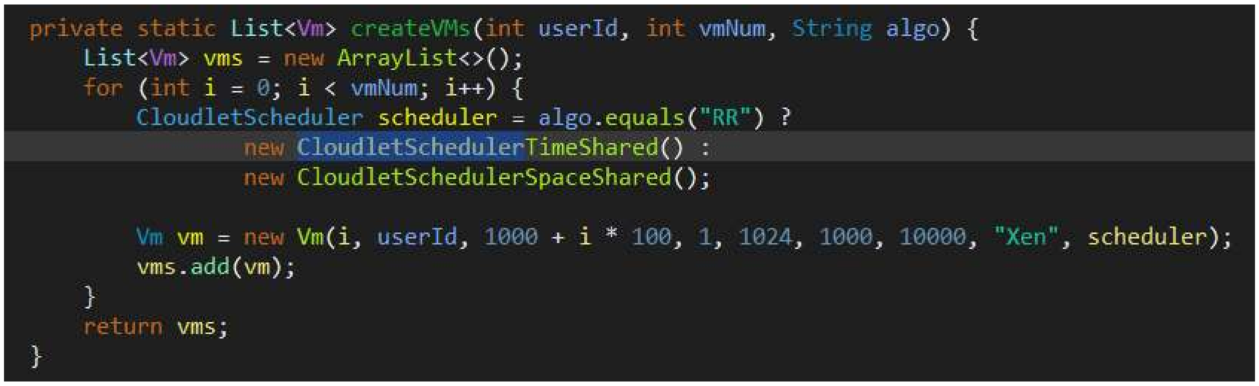
Implementation Steps of main code:

Step 1: Create Cloudlets



```
    private static List<Cloudlet> createCloudlets(int userId) {
        List<Cloudlet> cloudlets = new ArrayList<>();
        int[] lengths = {6000, 20000, 8000, 10000, 120000, 6000, 70000, 9000, 40000, 30000};
        for (int i = 0; i < CLOUDLET_NUM; i++) {
            int len = lengths[i % lengths.length];
            UtilizationModel util = new UtilizationModelStochastic();
            Cloudlet cl = new Cloudlet(i, len, 1, 300, 300, util, util, util);
            cl.setUserId(userId);
            cloudlets.add(cl);
        }
        return cloudlets;
    }
```

Step 2: Create VMs with Specific Schedulers

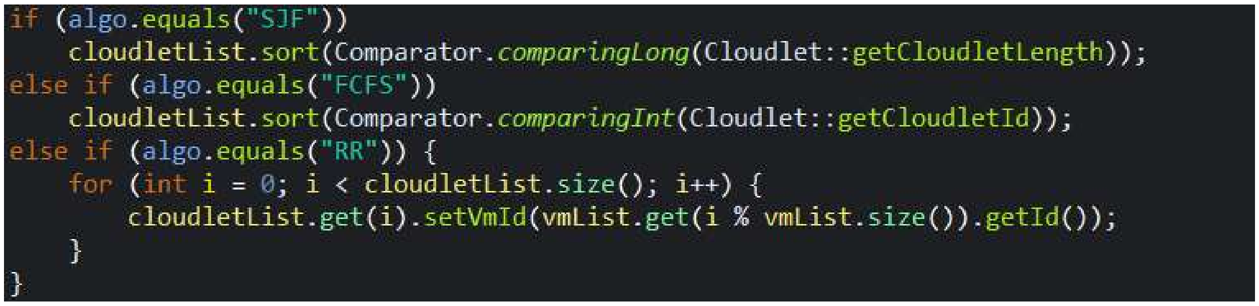


```
private static List<Vm> createVMs(int userId, int vmNum, String algo) {
    List<Vm> vms = new ArrayList<>();
    for (int i = 0; i < vmNum; i++) {
        CloudletScheduler scheduler = algo.equals("RR") ?
                new CloudletSchedulerTimeShared() :
                new CloudletSchedulerSpaceShared();

        Vm vm = new Vm(i, userId, 1000 + i * 100, 1, 1024, 1000, 10000, "Xen", scheduler);
        vms.add(vm);
    }
    return vms;
}
```

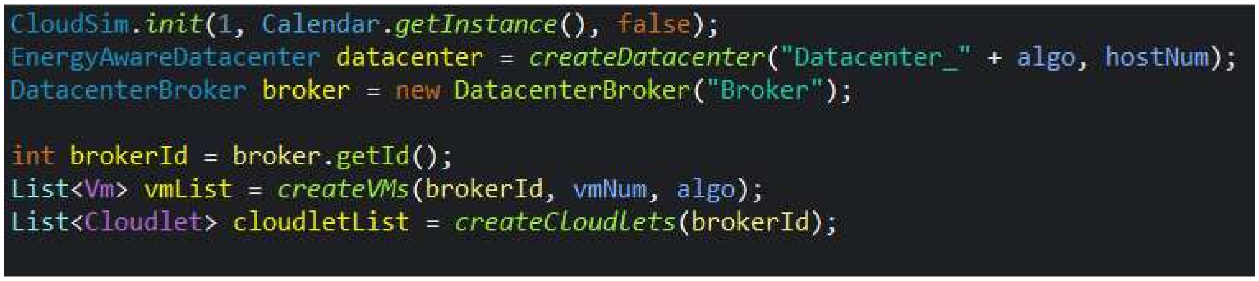
Step 3: Apply Scheduling Algorithm Logic

- **RR: Assign cloudlets to VMs in round-robin**
- **SJF: Sort cloudlets by length**
- **FCFS: Sort cloudlets by ID**



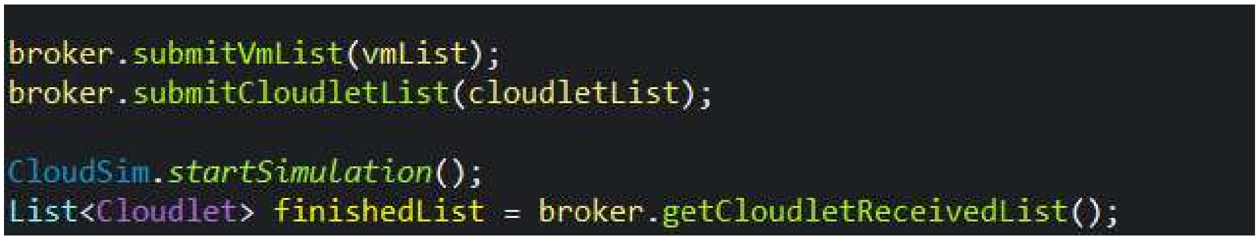
```java
if (algo.equals("SJF"))
    cloudletList.sort(Comparator.comparingLong(Cloudlet::getCloudletLength));
else if (algo.equals("FCFS"))
    cloudletList.sort(Comparator.comparingInt(Cloudlet::getCloudletId));
else if (algo.equals("RR")) {
    for (int i = 0; i < cloudletList.size(); i++) {
        cloudletList.get(i).setVmId(vmList.get(i % vmList.size()).getId());
    }
}
```

Step 6: Submit Lists to Broker and starts simulation



```java
CloudSim.init(1, Calendar.getInstance(), false);
EnergyAwareDatacenter datacenter = createDatacenter("Datacenter_" + algo, hostNum);
DatacenterBroker broker = new DatacenterBroker("Broker");

int brokerId = broker.getId();
List<Vm> vmList = createVMs(brokerId, vmNum, algo);
List<Cloudlet> cloudletList = createCloudlets(brokerId);
```



```java
broker.submitVmList(vmList);
broker.submitCloudletList(cloudletList);

CloudSim.startSimulation();
List<Cloudlet> finishedList = broker.getCloudletReceivedList();
```

Step 4: Extend Datacenter for Energy Profiling

LINEAR power model:

P = idle + (max - idle) * Utilization

Where, Utilization is CPU utilization scored from 0 to 1

Max:Peak power in watts

Min:Ideal power in watts

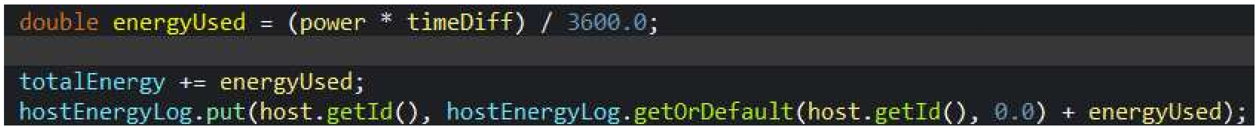
Formal Justification & Citations to the formula above

[1] A. Beloglazov, J. Abawajy, and R. Buyya,
"Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing,"
Future Generation Computer Systems, vol. 28, no. 5, pp. 755–768, May 2012.
doi: 10.1016/j.future.2011.04.017

[2] D. L. Bartlett and M. LaPre,
"Calculating Server Power Consumption,"
ITProToday, May 2009.
[Online].Available:https://www.itprotoday.com/microsoft-windows/calculating-server-power-costs

[3] M. Lewis, et al.,"Run-time Energy Consumption Estimation Based on Workload in Server Systems," *USENIX HotPower*, 2008.[Online].
Available:https://www.usenix.org/legacy/event/hotpower08/tech/full_papers/lewis/lewis_html/

<u>**Host Energy Tracking Logic:**</u>



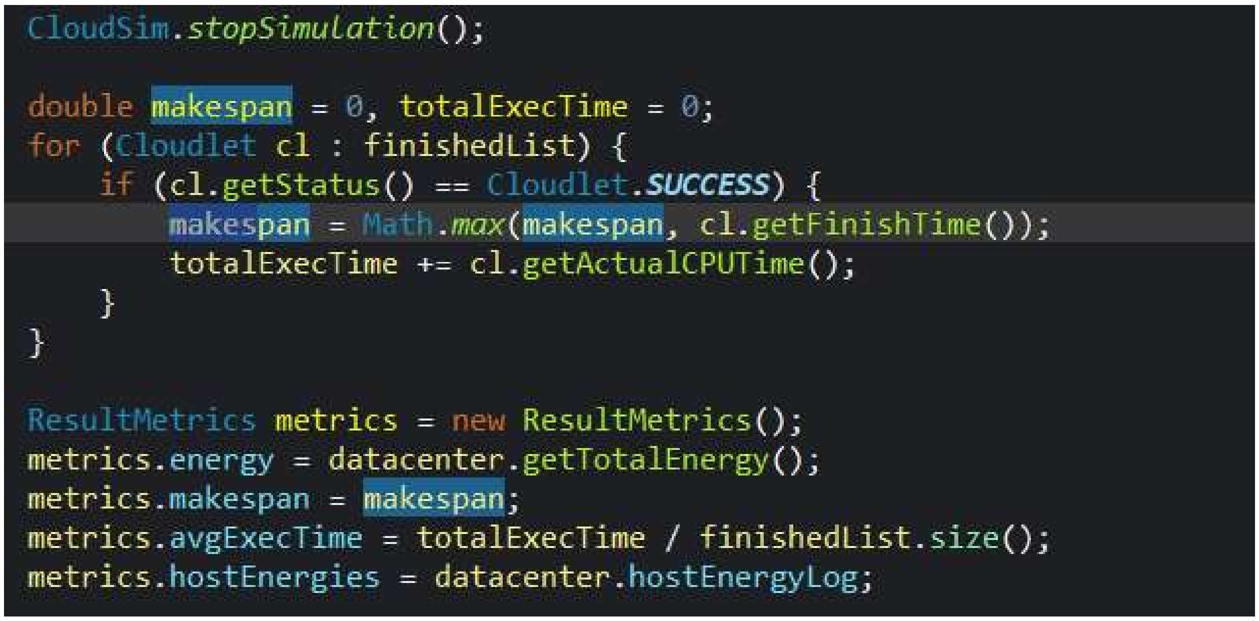
```
double energyUsed = (power * timeDiff) / 3600.0;

totalEnergy += energyUsed;
hostEnergyLog.put(host.getId(), hostEnergyLog.getOrDefault(host.getId(), 0.0) + energyUsed);
```

Step 7: Extract Results

Metrics calculated:

- **Total Energy (kWh)**
- **Makespan (s)**
- **Average Execution Time (s)**
- **Host-wise energy**

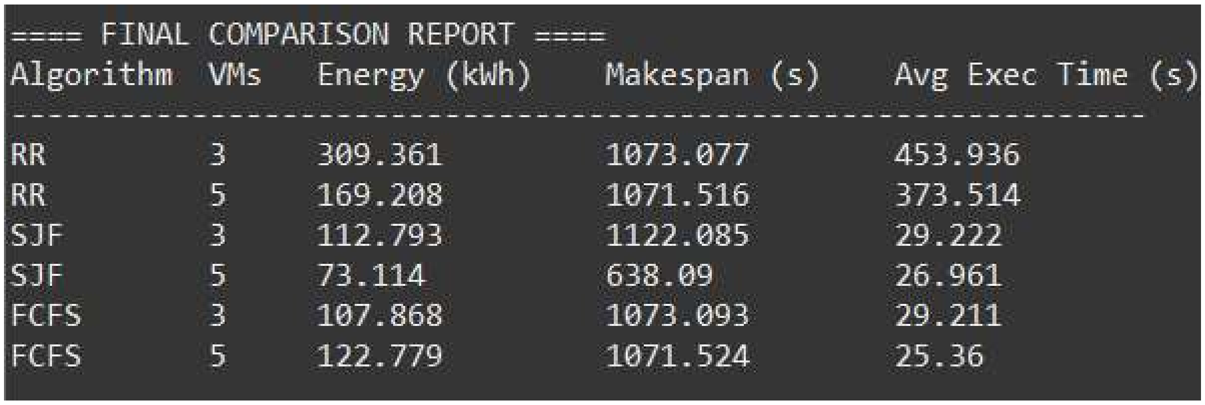


```
CloudSim.stopSimulation();

double makespan = 0, totalExecTime = 0;
for (Cloudlet cl : finishedList) {
    if (cl.getStatus() == Cloudlet.SUCCESS) {
        makespan = Math.max(makespan, cl.getFinishTime());
        totalExecTime += cl.getActualCPUTime();
    }
}

ResultMetrics metrics = new ResultMetrics();
metrics.energy = datacenter.getTotalEnergy();
metrics.makespan = makespan;
metrics.avgExecTime = totalExecTime / finishedList.size();
metrics.hostEnergies = datacenter.hostEnergyLog;
```

Final Comparison Report:



```
==== FINAL COMPARISON REPORT ====
Algorithm  VMs   Energy (kWh)    Makespan (s)    Avg Exec Time (s)
----------------------------------------------------------------
RR         3     309.361         1073.077        453.936
RR         5     169.208         1071.516        373.514
SJF        3     112.793         1122.085        29.222
SJF        5     73.114          638.09          26.961
FCFS       3     107.868         1073.093        29.211
FCFS       5     122.779         1071.524        25.36
```

Final Metrics Table Notes:

- **Energy is calculated using a linear power model.**
- **Makespan represents the time at which the last cloudlet finishes.**
- **Average Execution Time is computed over 100 cloudlets.**
- **All tests used the same cloudlet set and host configurations for fairness.**

CI/CD Pipeline Implementation

The CI/CD pipeline for this project consists of:

1. GitHub Actions Workflow - Automatically triggered on code changes to handle build and deployment
2. Build Automation - Maven compiles Java code and runs unit tests
3. Deployment Strategy - Automated deployment to development, staging, and production environments

Benefits of CI/CD Implementation

- Reduced manual deployment effort
- Consistent build and test environment

- Faster release cycles
- Better code quality through automated testing
- Enhanced team collaboration

Conclusion

This project successfully demonstrates the use of the CloudSim framework to model and evaluate the performance of different cloud scheduling algorithms—Round Robin (RR), Shortest Job First (SJF), and First Come First Serve (FCFS)—under realistic conditions. By simulating 100 cloudlets with varying lengths and applying a linear energy model, the simulation environment reflects practical data center behavior with regard to task execution and energy usage.

Key performance metrics such as total energy consumption, makespan, and average execution time were compared across algorithms and VM configurations. The results highlight that while SJF often achieves better execution efficiency due to prioritization of shorter tasks, RR and FCFS offer predictable and fair scheduling behavior. Each strategy presents trade-offs depending on system goals (e.g., energy saving vs. responsiveness).

In addition, a CI/CD pipeline using GitHub Actions, Maven, and JUnit was implemented to automate the build, test, and packaging processes. This improves maintainability, reduces manual errors, and ensures consistency in testing and deployment, adhering to modern DevOps practices.

References:

[1] R. Buyya, R. Ranjan, and R. N. Calheiros,
"Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities,"
Proceedings of the 7th High Performance Computing and Simulation Conference (HPCS 2009), Leipzig, Germany, June 2009.

2] SPEC Power Committee,
SPEC Power and Performance Methodology Version 2.0.3,
Standard Performance Evaluation Corporation (SPEC), May 2017.
[Online]. Available:
https://www.spec.org/power/docs/SPEC-Power_and_Performance_Methodology.pdf

[3] GitHub,
"GitHub Actions Documentation," GitHub.
[Online]. Available: https://docs.github.com/en/actions